Introduzione al linguaggio Java (2 puntata)

Ignazio Palmisano Oriana Licchelli Domenico Redavid
Slides at `http://www.di.uniba.it/~palmisano`

Dipartimento di Informatica
Università degli Studi di Bari

2005/2006

- JDBC: Java DataBase Connectivity
- É uno strato di astrazione sui database di qualunque genere; package java.sql
- Richiede librerie specifiche per i database da collegare (MySQL, SQL Server. . .)
- É incluso il driver per database ODBC

```
public class JDBC {
  public void test() {
    try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection conn = DriverManager.getConnection("jdbc:mysql://host/model? user=user&password=password");
      Statement st = conn.createStatement();
      ResultSet set = st.executeQuery("SELECT * FROM Tabella");
      while (set.next()) {
      }
      set.close();
      st.close();
    } catch (ClassNotFoundException e) {e.printStackTrace();
    } catch (SQLException e) { e.printStackTrace();
    }
  }
}
```

Connessioni HTTP: Socket

- `java.net.Socket`, `java.net.ServerSocket`
- Consente di creare stream per la comunicazione tra processi
- Si basa su un protocollo client/server

```
package slides;
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.net.Socket;
public class Sockets {
   public void test() throws UnknownHostException, IOException {
      Socket s = new Socket(InetAddress.getByName("www.yahoo.it"), 80);
      int i = s.getInputStream().read();
      s.getOutputStream().write(i);
      s.close();
   }
}
```

Applets

Le applets sono

- Piccole applicazioni a cui si accede su un server Internet
- Automaticamente installate
- Un'applet ha un accesso limitato alle risorse (sandbox)

```
package slides;
import java.awt.*;
import javax.swing.JApplet;
public class Applets extends JApplet {
    public void paint (Graphics g) {
     g.drawString("A Simple Applet", 20,20);
    }
}
```


```
<HTML>
  <HEAD><TITLE>Sample Applet </TITLE></HEAD>
  </BODY>
    <APPLET CODE="slides.Applets" WIDTH= 200 HEIGHT=100>
       Il tuo browser non supporta le applet
    </APPLET>
</BODY> </HTML>
```

Threads

- Un thread è un flusso di esecuzione del programma
- Ci possono essere più flussi di esecuzione in contemporanea
- Stesso spazio di indirizzamento
- Problemi di sincronizzazione

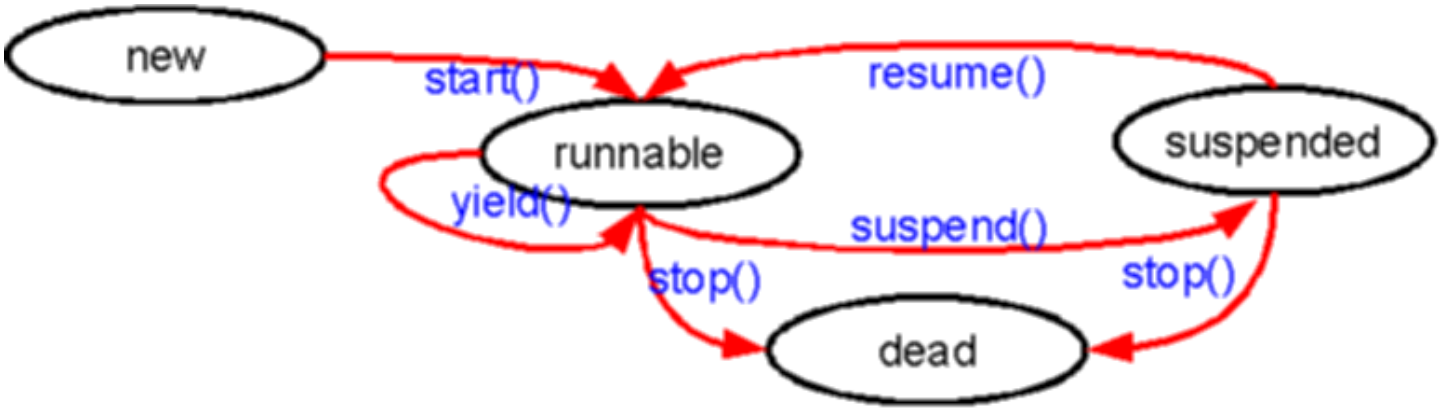
Creazione

- Implementare Runnable
- Ridefinire il metodo run()
- Avviare il thread con start()

Esempio

```
package slides;
class Contatore extends Thread implements Runnable {
   public void run() {
      int n = 0;
      while (true) {
         System.out.println(n);
         ++n;
      }
   }
   public static void main(String[] args) {new Contatore().start();}
}
```

Vita del thread



Thread: ciclo di vita

- Un thread appena creato è nello stato new; non è attivo
- Per attivare un thread, occorre chiamare start(); il thread è runnable
- Un thread runnable ottiene ogni tanto il processore
- Un thread runnable può cedere il passo agli altri con yield() rimanendo attivo
- Un thread attivo può andare in stato di suspended (non ottiene mai il processore) con suspend()
- Un thread sospeso ritorna in esecuzione con resume()
- Un thread può morire (e non ritornare in esecuzione) con stop()

Perché un'interfaccia Runnable?

- Distinzione tra interfaccia e implementazione
- Si vuole gestire come un thread qualcosa che deriva da altre

Sincronizzazione (1)

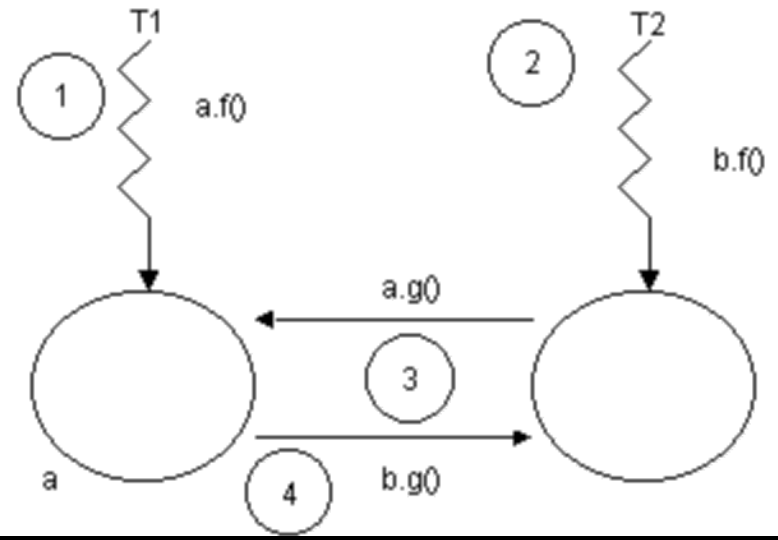
- Garantire la completa esecuzione dei metodi che eseguono funzionalità critiche
- Quando un thread accede ad un oggetto sincronizzato, lo blocca
- Ogni nuovo thread viene sospeso
- Quando un thread libera l'oggetto, si attiva un thread sospeso
- La sincronizzazione rallenta un programma
- Se si ridefinisce un metodo sincronizzato, il metodo ridefinito non è automaticamente sincronizzato

Sincronizzazione (2)

- wait() e notify(): metodi della classe Object
- wait() provoca la seguente situazione:
  - sospende il thread corrente e lo pone in attesa
  - un altro thread va in esecuzione
  - il thread viene riattivato non appena si esegue notify() sull'oggetto
- Un thread riattivato accede all'oggetto e riprende l'esecuzione dal punto in cui era stato interrotto
- notify() riattiva un solo thread, per riattivare tutti quelli in attesa si usa il metodo notifyAll()

Esempio

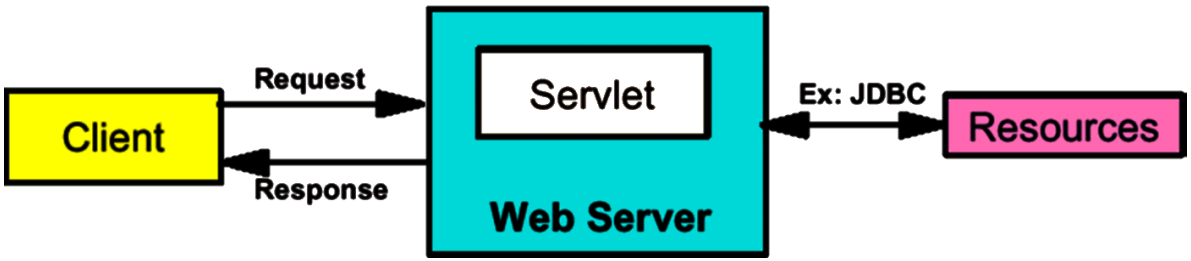
```
package slides;
public class Stack {
  private int top;
  private int[] stack;
  synchronized void push(int x)
                    throws InterruptedException {
    while (top > stack.length) {  wait();  }
  }
  synchronized int pop() {
    int r = stack[--top];
    notify();
    return r;
  }
}
```



- T1 esegue a.f() e blocca a
- T2 esegue b.f() e blocca b
- T1 chiama b.g() ma b è bloccato, e si sospende
- T2 chiama a.g() ma a è bloccato, e si sospende
- Stallo: T1 e T2 sono sospesi in eterno

Servlet

- Classi Java, eseguite lato server, independenti dalla piattaforma e dal protocollo
- Eseguite in un Application Server (IBM WebSphere Application Server, Apache Tomcat, ecc)
- Caricate nella Java Virtual Machine dell'Application Server
- Analoghe alle CGI, ma scritte in Java



Servlet e ciclo di vita

- HTTPServlet è la classe astratta che le servlet che lavorano con HTTP devono estendere
- void service(HttpServletRequest, HttpServletResponse) è il principale metodo da ridefinire
- Il package è javax.servlet.http
- Inizializzazione di una Servlet: il metodo init() con parametro la configurazione del Servlet (ServletConfig)
- Soddisfazione delle richieste: la richiesta è un oggetto HttpServletRequest; la risposta è un HttpServletResponse (pagina Html inviata al client)
- Distruzione della Servlet: il metodo destroy() viene richiamato quando l'Application Server viene chiuso o quando la servlet va modificata

Esempio

```
package slides;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleHttpServlet extends HttpServlet {
  protected void service(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException {
   res.setContentType("text/html");
   PrintWriter out = res.getWriter();
   out.println("<HTML><TITLE>SimpleHttpServlet</TITLE><BODY>");
   out.println("<H2>ServletAPI Example - SimpleHttpServlet </H2><HR>");
   out.println("<H4>This is about as simple a servlet as it gets!</H4>");
   out.println("</BODY><HTML>");
   out.close();
  }
}
```

HTTPSession

- HttpSession è una sessione tra client e server (stato) che persiste finché non viene esplicitamente chiusa oppure scade.
- Metodi utili per gestire la sessione:
  - getSession: ritorna la session corrente. Ha un parametro:
  - true: crea una nuova session se non esiste
  - false: ritorna null se la session non esiste
- setAttribute(nome, valore): permette di inserire un oggetto nella session corrente
- getAttribute(nome): permette di ritrovare un oggetto nella session corrente

Serializzazione

- Gli oggetti in sessione devono implementare l'interfaccia Serializable
- Un oggetto serializzabile può essere salvato in uno stream (ad esempio in un file) e poi ricreato a partire dallo stream
- La serializzazione nativa Java è binaria, ma si può costruirne una propria (ad esempio, in XML)
- Serializable non dichiara nessun metodo
- Sono serializzabili i tipi primitivi, le stringhe e tutti i tipi che comprendono solo questi tipi

Java Server Pages

- Java Server Pages: pagine dinamiche in cui il linguaggio di scripting è Java
- Una JSP ha bisogno di un Application Server
- Alla prima invocazione, la JSP viene compilata e viene generata una Servlet

Esempio Servlet

```
package slides;
import java.io.IOException;
public class Saluto extends HttpServlet {
  protected void service(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException {
    String nameLoc = req.getParameter("NAME");
    String output = "NESSUNO";
    if (nameLoc != null)
        output = nameLoc;
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><TITLE>Saluto</TITLE><BODY>");
    out.println("<H1>Hello " + output + "</H1>");
    out.println("</BODY></HTML>");
    out.close();
  }
}
```

Esempio JSP

Input

```
<html><head><title> Input per Saluto </title></head>
<body>
 <form name=input method=Post Action = "Saluto.jsp">
  <h2> Inserire il proprio nome: </h2>
  <input type="text" name="name">
  <input type="submit" value="OK">
 </form>
</body></html>
```

Saluto

```
<html><head><title> Saluto </title></head>
<body>
<% String chi = request.getParameter("name");
   if (chi==null) {chi="NESSUNO";} %>
<h1> Hello <%= chi %> </h1>
</body></html>
```

JSP e JavaBean

- Le JSP da sole non permettono una buona divisione tra codice java e codice html.
- Deleghiamo il saluto ad un Bean chiamato `salutoBean`
- Creazione di un Bean:
  - <jsp:useBean id="nome" class="classe" />
- Inizializzazione di una proprietà del Bean:
  - <jsp:setProperty name="nome" property="prop" param="valore_parametro" />
- Utilizzo di una proprietà del Bean:
  - <jsp:getProperty name="nome" property="prop" />
  - <%= nomeBean.metodo() %>

Esempio JavaBean

Esempio di bean:

```
package slides;
public class SalutoBean {
    private String chi;
    public SalutoBean() { super(); }
    public void setChi(String chi) { this.chi = chi; }
    public String getChi() { return chi; }
    public String getSaluto() {
  return "Hello " + ((chi==null) ? "NESSUNO" : chi);
    }
}
```

JSP corrispondente:

```
<html><head><title> Prova di una pagina JSP </title></head>
<body>
 <jsp:useBean id="saluta" class="slides.SalutoBean" />
 <jsp:setProperty name="saluta" property="chi" param="Pippo" />
 <%= saluta.getSaluto() %>
</body></html>
```

Direttive JSP

```
<html><head><title> Direttive </title></head>
<%@page language="java" %>
<%@page isErrorPage="true" %>
<%@page isThreadSafe="true" %>
<%@page import="saluto.jsp"%>
<%@page errorPage="exception.jsp"%>
<body>
 <% //questo è uno scriptlet %>
 <%="ciccio"%><!--questa è un'espressione
      che produce l'output di "ciccio"
      nella pagina-->
  <h2> Inserire il proprio nome: </h2>
  <input type="text" name="name">
  <input type="submit" value="OK">
 </form>
</body></html>
```

Oggetti impliciti

- Oggetti impliciti in JSP e Servlet:
- request: istanza di HttpServletRequest
  - getParameter(nomeAttributo)
  - Ciclo di vita = richiesta HTTP
- response: istanza di HttpServletResponse
  - Ciclo di vita = composizione risposta html
- session: istanza di HttpSession
  - disponibile se è stato settato a true l'attributo session della direttiva page
  - setAttribute(nome, valore)
  - getAttribute(nome)
  - Ciclo di vita = sessione

Gestione delle eccezioni nelle JSP

- Con l'attributo error-Page della direttiva page si può specificare la pagina di gestione dell'errore
- Una pagina di gestione delle eccezioni ha il valore true dell'attributo isErrorPage della direttiva page
- Quando viene sollevata un'eccezione in una JSP, il controllo passa alla relativa pagina di errore, che vede l'oggetto implicito exception che rappresenta l'eccezione sollevata

```
<%@ page language="java" isThreadSafe="false" %>
<%@ page isErrorPage="true" %>
<%@ page info = "Form per inserire il nome di un file" %>
<html> <head> <title> Error Page </title> </head>
<body>
 <h2>SI E' VERIFICATO UN ERRORE </h2>
 <p><%= exception.getMessage() %>
</body></html>
```

Esempio JSP: non thread safe - perché?

```
<%@ page language="java" isThreadSafe="false" %>
<%@ page info = "Contatore di accessi" %>
<%! int count=0;
    private static int FREQUENZA = 10;
    public void more() {count++;}%>
<html><head><title> Counter </title></head><body>
 <% more();
  // Se il numero è divisibile per FREQUENZA,
  // visualizziamo un messaggio
  if ((count % FREQUENZA)==0) { %>
  <%@ include file = "haivinto.jsp"%>
  <% } else { %>
  <%@ include file = "nonhaivinto.jsp"%>
  <% } %>
</body></html>
```

haivinto.jsp:

```
<h1> BRAVO HAI VINTO </h1>
perché sei il visitatore numero: <%= count %>
```

nonhaivinto.jsp:

```
<H1> NON HAI VINTO</h1>,
sei il visitatore numero: <%= count %>
```